wangtek 41 Moreland Road • Simi Valley, CA 93065 • (805) 583-5255 • Telex 650450

APPLICATION NOTES

NO. 3008-A

PC-36 SOFTWARE DRIVER

The attached listing of a Tape Driver Program can be used as an aid for programmers attempting to write "Software Driver Programs" for WANGTEK PC-36 Controller Boards. The PC-36 Controller is designed to interface the basic quarter inch tape drive to IBM-PC, IBM-XT, IBM-AT computers and their compatibles.

In addition, the attached Driver Program is useful in writing software driver programs to support other operating systems, such as Xenix, Unix, CP/M-86 etc.

There are three main subroutines called in by the Software Driver (TDRIVER.C) Program. They are "Writedata", "Readdata", and "Initialize".

WRITEDATA - The following commands take place in this subroutine:

1. Reset
2. Read Status
3. Rewind to BOT
4. Write
5. Write filemark
6. End Write (This drops "Online" and rewinds the tape to BOT)

READDATA - The following commands take place in this subroutine:

1. Reset
2. Read Status
3. Rewind to BOT
4. Read
5. End Read

INTIALIZE

This subroutine initializes the interrupt vector table to interrupt request address three. Anytime any exception is asserted by the drive, the program will read status, display the status message and then exit the program.

```
.cw8
.lh7
/*************************************************************************/
/** Program name:        TDRIVER.C                                      **/
/** Author:              Tony Sotery                                    **/
/** Creation date:       08/03/1985                                     **/
/** Description:         This program will perform as a driver program. **/
/**                      The purpose of the program is to perform tape  **/
/**                      drive operation and data transfer To and From  **/
/**                      the tape.                                      **/
/** Revision History:    Version 1.00                                   **/
/*************************************************************************/
#include <stdio.h>

/*-----------------------------------------------------------------------*/
/* The definition below provides the tape drive exception status         */
/*-----------------------------------------------------------------------*/
#define FILEMARK     0X0001
#define BIENL        0X0002
#define UDE          0X0004
#define EOT          0X0008
#define WRITE_PROT   0X0010
#define NOT_ON_LINE  0X0020
#define NO_CARTRIDGE 0X0040
#define EX0          0X0080
#define RESET        0X0100
#define BOT          0X0800
#define NO_DATA      0X2000
#define ILLEGAL_CMD  0X4000
#define EX1          0X8000

#define BLKSIZE 512             /* each block is 512 bytes */
#define BUFSIZE 16 * BLKSIZE    /* buffer size is 8 kbytes */

#define STATUSREG 0x300         /* Status register address */
#define READY          0x01     /* ready bit define        */
#define EXCEP          0x02     /* exception bit define    */

char buf[BUFSIZE];              /* the buffer to used      */

main()
{
initialize();                   /* initialize the program by setting the interrupt */
writedata();                    /* write data from the memory to the tape */
readdata();                     /* read data from the memory to the tape  */
}
/*************************************************************************/
/** Routine name:        writedata                                      **/
/** Description:         write data from the memory buffer to the tape  **/
/** Author:              Tony Sotery                                    **/
/** Date:                08/06/1985                                     **/
/** Called by:           main                                           **/
/** Calls:               Assembly code or error reporting routines      **/
/*************************************************************************/
writedata()
{
int srb[3];                     /* three status word register array */
int i;

if (t_reset())                  /* reset the tape drive */
  error(1);                    /* report command did not go through */
if (rdstatus(srb))              /* read the status registers */
  error(2);
                                /* report any of the following errors */
reperr(srb[0],FILEMARK | BIENL | UDE | EOT | WRITE_PROT | NOT_ON_LINE
                 | NO CARTRIDGE | NO DATA | ILLEGAL CMD);
```

```
if (rwind())                   /* rewind the tape to BOT */
  error(3);
if (rdyexc())
  {
  if (rdstatus(srb))
    error(2);
  reperr(srb[0],FILEMARK | BIENL | UDE | EOT | WRITE_PROT | NOT_ON_LINE
                | NO_CARTRIDGE | RESET | NO_DATA | ILLEGAL_CMD);
  }
printf("Writing Data To Tape\n");

if (wstart())                  /* start write operation */
  error(4);
for (i=0;i<100;i++)            /* write 1600 blocks */
  {
   wtape(buf,16);              /* perform the write operation */
   if (rdyexc())
     {
     if (rdstatus(srb))
       error(2);
     reperr(srb[0],FILEMARK | BIENL | UDE | EOT | WRITE_PROT | NOT_ON_LINE
                | NO_CARTRIDGE | RESET | BOT | NO_DATA | ILLEGAL_CMD);
     }
  }
if (wmark())                   /* write file mark at the end of data written */
  error(6);
if (rdyexc())
  {
  if (rdstatus(srb))
    error(2);
  reperr(srb[0],FILEMARK | BIENL | UDE | EOT | WRITE_PROT | NOT_ON_LINE
                | NO_CARTRIDGE | RESET | BOT | NO_DATA | ILLEGAL_CMD);
  }
if (wend())                    /* end the write operation by dropping online */
  error(5);
if (rdyexc())                  /* report if any error occured */
  {
  if (rdstatus(srb))
    error(2);
  reperr(srb[0],FILEMARK | BIENL | UDE | EOT | WRITE_PROT | NOT_ON_LINE
                | NO_CARTRIDGE | RESET | BOT | NO_DATA | ILLEGAL_CMD);
  }
}
/*****************************************************************************/
/** Routine name:          readdata                                        **/
/** Description:           read data from the tape to the buffer           **/
/** Author:                Tony Sotery                                     **/
/** Date:                  08/05/1985                                      **/
/** Called by:             main                                            **/
/** Calls:                 Assembly code or error reporting routines       **/
/*****************************************************************************/
readdata()
{
int srb[3];                    /* status bytes register holder */
int i;

if (t_reset())                 /* reset the tape drive */
  error(1);
if (rdstatus(srb))             /* read status to clear reset exception */
  error(2);
                               /* report the error condition */
reperr(srb[0],FILEMARK | BIENL | UDE | EOT | NOT_ON_LINE
                | NO_CARTRIDGE | NO_DATA | ILLEGAL_CMD);
if (rwind())                   /* rewind to the beginning of the tape */
  error(3);
if (rdyexc())
```

```
    {
    if (rdstatus(srb))
      error(2);
    reperr(srb[0],FILEMARK | BIENL | UDE | EOT | NOT_ON_LINE
                   | NO_CARTRIDGE | RESET | NO_DATA | ILLEGAL_CMD);
    }
  printf("Reading Data From Tape\n");
  if (rstart())                    /* start the read operation */
    error(7);
  for (i=0;i<100;i++)              /* read 1600 blocks */
     {
     rtape(buf,16);                /* perform the read operation */
     if (rdyexc())
       {
       if (rdstatus(srb))
         error(2);
       reperr(srb[0],FILEMARK | BIENL | UDE | EOT | NOT_ON_LINE
                   | NO_CARTRIDGE | RESET | BOT | NO_DATA | ILLEGAL_CMD);
       }
     }
  if (rend())                      /* end the read operation by dropping online */
    error(8);
  if (rdyexc())
    {
    if (rdstatus(srb))
      error(2);
    reperr(srb[0],FILEMARK | BIENL | UDE | EOT | NOT_ON_LINE
                  | NO_CARTRIDGE | RESET | BOT | NO_DATA | ILLEGAL_CMD);
    }
  }
/*************************************************************************/
/** Routine name:      reperr                                          **/
/** Description:       report an error if any and exit                 **/
/**                    .                                               **/
/**                    .                                               **/
/** Author:            Tony Sotery                                     **/
/** Date:              3/5/85                                          **/
/** Parameters:        srb                                             **/
/*************************************************************************/
reperr(srb0,s)
int srb0,s;
{
int i,s0;
s0=srb0 & s;                     /* find out which exception to report */
/*---------------------------------------------------------------------*/
/* report the given error if they occured                              */
/*---------------------------------------------------------------------*/
if (s0)
  {
  i=22;
  if (s0 & NOT_ON_LINE)
    printf("Drive not online\n");
  if (s0 & NO_CARTRIDGE)
    printf("No cartridge\n");
  if (s0 & WRITE_PROT)
    printf("Tape is write protected\n");
  if (s0 & FILEMARK)
    printf("Filemark detected\n");
  if (s0 & BIENL)
    printf("Block in error not located\n");
  if (s0 & UDE)
    printf("Unrecoverable data error\n");
  if (s0 & EOT)
    printf("End of tape\n");
  if (s0 & NO_DATA)
    printf("No data detected\n");
  if (s0 & RESET)
    printf("Reset occured\n");
  if (s0 & BOT)
    printf("Begining of tape\n");
  if (s0 & ILLEGAL_CMD)
    printf("Illegal command\n");

  exit();                             /* exit the program */
  }
}
/***************************************************************************/
/** Routine name:     error                                              **/
/** Description:      print a message showing the error and exit         **/
/** Author:           Tony Sotery                                        **/
/** Date:             2/29/85                                            **/
/** Parameters:       num: number of error that occured                  **/
/***************************************************************************/
error(num)
int num;
{
printf("Command did not go through [%d]",num);
exit();
}
/***************************************************************************/
/** Routine name:     rdyexc                                             **/
/** Description:      wait for ready or exception and return the status  **/
/** Author:           Tony Sotery                                        **/
/** Date:             2/29/85                                            **/
/** Parameters:                                                          **/
/***************************************************************************/
rdyexc()
{
int s;
for (;;)                              /* loop until ready or exception */
  {
  s=(inportb(STATUSREG) & 0xff);      /* read the status register */
  if (!(s & EXCEP))                  /* check if exception have occured */
    break;
  if (!(s & READY))                   /* check if controller is ready */
    break;
  }
  return(!(s & EXCEP));               /* return exception if it occured */
}
/***************************************************************************/
/** Routine name:  INITIALIZE                                            **/
/** Description:   Perform required program initialization               **/
/***************************************************************************/
initialize()
{
unsigned int isr();
unsigned int extraseg,dataseg,codeseg,offseg;
struct {unsigned int cs,ss,ds,es;} rrv;

segread(&rrv.cs);                     /* get the segment value */
extraseg = rrv.es ;
dataseg = rrv.ds ;
codeseg= rrv.cs;

isrinit();                            /* save our "DS" in code segment of "ISR" */
outportb(0x21,(inportb(0x21) & 0xf7)); /* enable irq3 interrupt for the PC-36 controler */
offseg=isr;                           /* get the offset for interrupt service routine */
pokew(0x2c,0,offseg);                 /* set interrupt vector to interrupt service routine */
pokew(0x2e,0,codeseg);
}
```

```
;*****************************************************************************
;**                                                                         **
;**                              TULIB.DEF                                  **
;**                                                                         **
;**  This file contains all the declaration and defines for file TULIB.ASM  **
;**  and TULIB1.ASM                                                         **
;**                                                                         **
;*****************************************************************************
statport        equ 300h        ;status port
ctlport         equ 300h        ;control port
dataport        equ 301h        ;data port
cmdport         equ 301h        ;command port
;
ready           equ 1           ;ready bit
excep           equ 2           ;exception bit
dirc            equ 4           ;direction bit
online          equ 1           ;online command
reset           equ 2           ;reset command
request         equ 4           ;request command
request_off     equ 0fbh        ;request command off
xfer            equ 10h         ;xfer command
cmdoff          equ 0           ;turns off command
;
rddata          equ 080h        ;read data
readfm          equ 0a0h        ;read file mark
wrtdata         equ 040h        ;write data
writefm         equ 060h        ;write file mark
rdstat          equ 0c0h        ;read status command
position        equ 020h        ;position command
bot             equ 01h         ;  rewind to bot
erase           equ 02h         ;  erase tape
retention       equ 04h         ;  retention tape
;
eqdma           equ 8h          ;enable dma command
;                                 8h=ch1 or ch2
;                                 10h=ch3
chan            equ 1           ;dma channel no.
addreg          equ 02h
wcreg           equ 03h
pagereg         equ 83h         ;ch1=83h, ch3=82h ,ch2=81h
cmdreg          equ 08h
statusreg       equ 08h
maskreg         equ 0ah
modereg         equ 0bh
clearff         equ 0ch
dma_write       equ 48h+chan
dma_read        equ 44h+chan
;
wci     dw      ?
blksize equ     512     ;block size
;
fail    equ     1
success equ     0
dma_rdy equ     0
not_rdy equ     2
;
sbs     struc           ;structure to hold six status bytes, return
old_bp  dw      ?       ;old bp
retaddl dw      ?       ;return address
sb1     dw      ?       ;status byte1
sb2     dw      ?       ;status byte2
sb3     dw      ?       ;status byte3
sb4     dw      ?       ;status byte4
sb5     dw      ?       ;status byte5
sb6     dw      ?       ;status byte6
sbs     ends
```

```
;
args    struc              ;structure to hold the parameters that are
od_bp   dw      ?          ;being passed. Old bp
retadl  dw      ?          ;return address
arg1    dw      ?
arg2    dw      ?
arg3    dw      ?
arg4    dw      ?
arg5    dw      ?
arg6    dw      ?
arg7    dw      ?
arg8    dw      ?
args    ends
;
@code   ends
@datab  segment

mbits           db      ?       ; map bits into each other
numblock        dw      ?       ; numblock to read or write
exceptio        dw      ?       ; exception variable
mode            dw      ?       ; dma operation mode
bufptr          dw      ?       ; used as buffer pointer to data buffers
stack           dw      0       ; initialize a stack
;
@datab  ends
```

```
;**********************************************************************/
;** Program Name:        TULIB.ASM                                  **/
;** Author:              Tony Sotery                                **/
;** Creation date:       12/14/84                                   **/
;** Description:         Tape drive controller command library      **/
;**                      This module contains a libray of all the command**/
;**                      that can be sent to the tape drive.        **/
;** Called by:           The "C" program.                           **/
;** Calls:               .                                          **/
;** Revision History:    Version 2.00                               **/
;**********************************************************************/
include         \c86\models.h
include         \c86\prologue.h
include         tulib.def

@code   segment byte public 'code'
        public  rstart  ;start read
        public  rend    ;end read
        public  rmark   ;read file mark
        public  wstart  ;start write
        public  wend    ;end write
        public  wmark   ;write file mark
        public  tension ;re-tension tape
        public  rwind   ;rewinds tape
        public  t_erase ;erase tape
        public  rdstatus;reads status
        public  t_reset ;reset
;
;****************************************
;  start read - c function
;
;       rstart()

rstart  proc    near
        mov     dx,statport     ;wait for ready
rdex:   in      al,dx
        test    al,excep        ;chk exception
        jz      r_ab            ;end the proc
        test    al,ready        ;is it ready
        jnz     rdex            ;loop if it is not ready
        mov     dx,cmdport      ;get the command port address
        mov     al,rddata       ;get the command
        out     dx,al           ;output the command to the port
        mov     dx,ctlport      ;get control port address
        mov     al,online       ;set online
        mov     mbits,al
        out     dx,al           ;send online
        call    sendcmd         ;send the command to the formatter
        mov     ax,success      ;operation was successful
        ret                     ;retuen to the caller
r_ab:                           ;abort operation
        mov     ax,fail         ;operation failed
        ret                     ;retun to the caller
rstart  endp

;****************************************
;  end read - c function
;
;       rend()
;
rend    proc    near
        mov     dx,statport     ;wait for ready
red:    in      al,dx
        test    al,excep        ;chk exception
        jz      nab             ;end the proc
        test    al,ready        ;is it ready
        jnz     red             ;loop if it is not ready

        mov     bx,success
rret:   mov     dx,ctlport      ;reset online
        mov     al,cmdoff
        out     dx,al
        mov     ax,bx           ;return code
        ret
nab:
        mov     bx,fail
        jmp     rret
rend    endp

;****************************************
;  read file mark - c function
;
;       rmark()
;
rmark   proc    near
        mov     dx,statport     ;wait for ready
rm:     in      al,dx
        test    al,excep        ;chk exception
        jz      m_ab             ;end the proc
        test    al,ready        ;is it ready
        jnz     rm              ;loop if it is not ready
        mov     dx,cmdport      ;read mark cmd
        mov     al,readfm
        out     dx,al
        mov     dx,ctlport      ;set online
        mov     al,online
        mov     mbits,al
        out     dx,al
        call    sendcmd
        mov     dx,statport
rn:
        in      al,dx
        test    al,excep
        jnz     rn
        mov     ax,success
        ret
m_ab:
        mov     ax,fail
        ret
rmark   endp
;
;****************************************
;  start write - c function
;
;       wstart()
;
wstart  proc    near
        mov     dx,statport     ;wait for ready
wd:     in      al,dx
        test    al,excep        ;chk exception
        jz      w_ab             ;end the proc
        test    al,ready        ;is it ready
        jnz     wd              ;loop if it is not ready
        mov     dx,cmdport      ;write data cmd
        mov     al,wrtdata
        out     dx,al
        mov     dx,ctlport      ;set online
        mov     al,online
        mov     mbits,al
        out     dx,al
        call    sendcmd
        mov     ax,success
        ret
w_ab:   mov     ax,fail
        ret
wstart  endp

;*****************************************
;  end write - c function
;
;       wend()
;
wend    proc    near
        mov     dx,statport     ;wait for ready
ee:     in      al,dx
        test    al,excep        ;chk exception
        jz      e_err           ;end the proc
        test    al,ready        ;is it ready
        jnz     ee              ;loop if it is not ready

        mov     bx,success
eret:   mov     dx,ctlport      ;reset online
        mov     al,cmdoff
        out     dx,al
        mov     al,4+chan       ;disable dma
        out     maskreg,al
        out     clearff,al

        mov     ax,bx           ;return code
        ret
e_err:
        mov     bx,fail
        jmp     eret
wend    endp

;*****************************************
;  write file mark - c function
;
;       wmark()
;
wmark   proc    near
        mov     dx,statport     ;wait for ready
wm:     in      al,dx
        test    al,excep        ;chk exception
        jz      wm_ab           ;end the proc
        test    al,ready        ;is it ready
        jnz     wm              ;loop if it is not ready
        mov     dx,cmdport      ;write mark cmd
        mov     al,writefm
        out     dx,al
        mov     dx,ctlport      ;set online
        mov     al,online
        mov     mbits,al
        out     dx,al
        call    sendcmd
        mov     dx,statport
wn:
        in      al,dx
        test    al,ready
        jnz     wn
        mov     ax,success
        ret
wm_ab:
        mov     ax,fail
        ret
wmark   endp

;*****************************************
```

```
;  rewinds tape - c function
;
;       rwind()
;
rwind   proc    near
        mov     mbits,0
        mov     dx,statport     ;wait for ready
wiw:    in      al,dx
        test    al,excep        ;chk exception
        jz      wi_ab           ;end the proc
        test    al,ready        ;is it ready
        jnz     wiw             ;loop if it is not ready
        mov     dx,cmdport      ;rewind cmd
        mov     al,position+bot
        out     dx,al
        call    sendcmd
        mov     ax,success
        ret
wi_ab:
        mov     ax,fail
        ret
rwind   endp

;****************************************
;  tensions tape - c function
;
;       tension()
;
tension proc    near
        mov     mbits,0
        mov     dx,statport     ;wait for ready
tiw:    in      al,dx
        test    al,excep        ;chk exception
        jz      ti_ab           ;end the proc
        test    al,ready        ;is it ready
        jnz     tiw             ;loop if it is not ready
        mov     dx,cmdport      ;tension cmd
        mov     al,position+retention
        out     dx,al
        call    sendcmd
        mov     ax,success
        ret
ti_ab:
        mov     ax,fail
        ret
tension endp

;****************************************
;  erases tape - c function
;
;       t_erase()
;
t_erase proc    near
        mov     mbits,0
        mov     dx,statport     ;wait for ready
eiw:    in      al,dx
        test    al,excep        ;chk exception
        jz      ei_ab           ;end the proc
        test    al,ready        ;is it ready
        jnz     eiw             ;loop if it is not ready
        mov     dx,cmdport      ;erase cmd
        mov     al,position+erase
        out     dx,al
        call    sendcmd
        mov     ax,success
        ret
```

```
ei_ab:
        mov     ax,fail
        ret
:_erase endp

;*****************************************
;  reads status - c function
;
;       rdstatus(srb)
;       int srb[3]      returns 6 status bytes
;
rdstatus proc   near
        push    bp
        mov     bp,sp
        mov     si,[bp].sbl

        mov     dx,statport     ;wait for ready
stwa:   in      al,dx
        test    al,excep        ;chk exception
        jz      stok            ;end the proc
        test    al,ready        ;is it ready
        jnz     stwa            ;loop if it is not ready

stok:
        mov     dx,cmdport      ;status command
        mov     al,rdstat
        out     dx,al
        call    sendcmd
        mov     cx,6            ;get 6 bytes
nxt_stat:
sr:
        mov     dx,statport
        in      al,dx
        test    al,excep
        jz      wi_sr
        test    al,ready
        jnz     sr
        push    cx
        mov     dx,dataport     ;read stat byte
        in      al,dx
        mov     [si],al
        inc     si
        mov     dx,ctlport      ;set request
        mov     al,request
        or      al,mbits
        out     dx,al
        mov     dx,statport
se:
        in      al,dx
        test    al,ready
        jz      se
        mov     cx,80h          ;wait >20us
sq:
        loop    sq
        mov     dx,ctlport      ;reset request
        mov     al,request_off
        and     al,mbits
        out     dx,al
        mov     dx,statport     ;next status
        pop     cx
        loop    nxt_stat
        mov     ax,success
        pop     bp
        ret
wi_sr:
        mov     ax,fail
        pop     bp
        ret
rdstatus endp

;****************************************
;  reset tape unit - c function
;
;       t_reset()
;
t_reset proc    near
        mov     dx,ctlport      ;reset
        mov     al,reset
        out     dx,al
        mov     cx,1000h        ;delay
dr:     loop    dr
        mov     al,cmdoff       ;un-reset
        out     dx,al
        mov     ax,success
        ret
t_reset endp

;****************************************
;  performs handshake to send command
;
;  destroys al,dx
;
sendcmd proc    near
        mov     dx,ctlport      ;set request
        mov     al,request
        or      al,mbits
        out     dx,al
        mov     dx,statport     ;wait ready
sw:     in      al,dx
        test    al,ready
        jnz     sw
        mov     dx,ctlport      ;reset request
        mov     al,request_off
        and     al,mbits
        out     dx,al
        mov     dx,statport     ;wait not ready
sn:     in      al,dx
        test    al,ready
        jz      sn
        ret
sendcmd endp

;****************************************
        include \c86\epilogue.h
        end
```

```
;************************************************************************/
;** Program Name:        TULIB1.ASM                                     **/
;** Author:              Tony Sotery                                    **/
,** Creation date:       12/14/84                                       **/
;** Description:         This module contains four major procedures:   **/
;**                      RTAPE: Read x number of blocks from the tape   **/
;**                      drive and save it in the address given in ds:bx **/
;**                      Wtape: Write x number of blocks from the memory **/
;**                      buffer addressed by ds:bx.                     **/
;**                      ISRINIT: Is used to get the "C" program's "DS". **/
;**                      ISR: Is the interrupt service routine, it sets **/
;**                      up the dma and starts dma.                     **/
;** Called by:           RTAPE: Must be called by the "C" program.      **/
;**                      WTAPE: Must be called by the "C" program.      **/
;**                      ISRINIT must be called by the "C" program.     **/
;**                      ISR is interrupt driven.                       **/
;**                      .                                              **/
;**                      .                                              **/
;** Revision History:    Version 2.00                                   **/
;************************************************************************/
include         \c86\models.h
include         \c86\prologue.h
include         tulib.def

@code   segment byte public 'code'
        public  rtape   ;read x blocks
        public  wtape   ;write x blocks
        public  isr
        public  isrinit
;
;*****************************************
;  read x block - c function
;
;       rtape(buffer,blkcount)
;
rtape   proc    near
        push    bp
        mov     bp,sp
        mov     ax,[bp].arg1    ; buffer area for the data transfered
        mov     bufptr,ax       ; set the pointer to the buffer area
        mov     ax,[bp].arg2    ; number of blocks to be transfered
        mov     numblock,ax
        mov     ax,0
        mov     wci,0           ; clear wci (word count interupt)
        mov     ax,dma_read     ; set the dma mode to read
        mov     mode,ax
        call    rdyexc          ; check if ready or exception have occured
        test    al,excep        ; if exception then done
        jz      r_done
        mov     bx,bufptr       ; set the address for the dma
        call    dma             ; start the dma
rloop2: cmp     exceptio,1      ; wait for either exception
        jz      r_done
        cmp     wci,1           ; or wci interrupt
        jnz     rloop2
r_done:
        pop     bp
        ret
rtape   endp

;*****************************************
;  write block - C function
;
;       wtape(buffer,blkcount)
;       char *buffer /* segment addr */
;       int blkcount /* number of block to wri
```

```
;
;
wtape   proc    near
        push    bp
        mov     bp,sp
        mov     ax,[bp].arg1        ; get the buffer address
        mov     bufptr,ax           ; set the buffer address
        mov     ax,[bp].arg2        ; get the number of blocks
        mov     numblock,ax         ; set the number of blocks
        mov     ax,0
        mov     wci,ax              ; clear the wci
        mov     ax,dma_write        ; set the dma mode to write
        mov     mode,ax
        call    rdyexc
        test    al,excep
        jz      w_done              ; chk exception
        mov     bx,bufptr           ; set the dma buffer address
        call    dma                 ; set the dma and started
lop2:   cmp     exceptio,1          ; wait for exception or,
        jz      w_done
        cmp     wci,1               ; wci from the ISR
        jnz     lop2
w_done:
        pop     bp
        ret
wtape   endp
;
;**********************************
;dma: set up dma address
;      and transfer 512 bytes
;
; ds:bx = transfer address
; destroys ax,cx,dx
dma     proc    near
        push    es
        push    cx
        cli
        mov     ax,mode
        out     clearff,al          ; clear first/last f/f, so lower and upper
        jmp     $+2
        out     modereg,al          ; output the mode byte
        mov     ax,ds
        mov     es,ax
        mov     ax,es               ; get current segment address
        mov     cl,4                ; multiply by 16
        rol     ax,cl
        mov     ch,al
        and     al,0f0h             ; zero out the low four bits
        add     ax,bx
        jnc     j33                 ; if addition produce carray, inc page reg.
        inc     ch
j33:
        push    ax
        out     addreg,al           ; output low address
        jmp     $+2
        mov     al,ah               ; output high address
        out     addreg,al
        mov     al,ch
        jmp     $+2
        and     al,0fh
        out     pagereg,al          ;output high 4 bits to the page reg
;determine count

        pop     ax
        add     ax,511
        mov     ax,511
        out     wcreg,al        ;output low byte of count
        jmp     $+2
        mov     al,ah
        out     wcreg,al        ;output high byte of count
        sti
        pop     cx
        pop     es
        mov     dx,ctlport
        mov     al,eqdma+online
        out     dx,al           ;inform host to enable dma on chan 1.
        mov     al,1            ;enable channel 1 command to dma
        out     maskreg,al      ;start dma
        ret
dma     endp

;***************************************************************************;
; The following routine saves the calling "C" program's "DS" for the interrupt;
; service routine use.  It is used for accessing "C" global variable from    ;
; an assembler routine.  This procedure must be called during initialization. ;
;***************************************************************************;
isrinit proc    near
        mov     ax,ds
        mov     cs:our_ds,ax
        ret
isrinit endp

our_ds  dw      0               ; local variable for storing caller's "DS".
save_ds dw      0               ; save the "DS" of whoever we have interrupted.
save_ss dw      0
save_sp dw      0
temp_ds dw      0
block   dw      0

isr     proc    near
        push    bp

        mov     cs:save_ds,ds   ; save the "DS" of whoever was interrupted.
        mov     ds,cs:our_ds    ; load the C program's "DS" into ours.
        mov     cs:save_ss,ss   ; save the stack informations.
        mov     cs:save_sp,sp
        mov     temp_ds,ds      ; set stack to data segment.
        mov     ss,temp_ds
        mov     sp,stack
        push    ax
        push    bx
        push    cx
        push    dx

        sti
        mov     al,20h          ; send eoi
        out     20h,al
        mov     dx,statport
        in      al,dx
        test    al,excep        ; test for exeception
        jz      excexit         ; if exeception then exit
        mov     exceptio,0      ; else no exception
        mov     dx,ctlport      ; disable dma on the everex board
        mov     al,online
        out     dx,al
        mov     cx,cs:block     ; block is for tracking number of blocks transfer
        inc     cx              ;   since last wci
        mov     cs:block,cx
        cmp     cx,numblock
        je      setwci          ; if blocks transfer equals the intended set wci and exit
        jmp     setdma          ; else setup next dma cycle
excexit:
        mov     cx,cs:block
        inc     cx
        cmp     cx,numblock
        jne     exit
        mov     wci,1
exit:
        mov     exceptio,1      ; exception true
        mov     cs:block,0
        jmp     done
setwci:
        mov     cs:block,0      ; clear block counter
        mov     wci,1           ; set wci true
        mov     al,4+chan       ; disable dma on the 8237
        out     maskreg,al
        jmp     done
setdma:
        add     bufptr,blksize  ; increment the buffer pointer
        mov     bx,bufptr
        call    dma

done:   pop     dx
        pop     cx
        pop     bx
        pop     ax
        mov     ss,cs:save_ss
        mov     sp,cs:save_sp
        mov     ds,cs:save_ds

        pop     bp
        iret
isr     endp

;*********************************************
;       rdyexc  wait for ready or exception
;
;
rdyexc  proc    near
        mov     dx,statport     ;wait for ready
rdex:   in      al,dx
        test    al,excep        ;chk exception
        jz      erdex           ;end the proc
        test    al,ready        ;is it ready
        jnz     rdex            ;loop if it is not ready
erdex:  ret                     ;return to the caller
rdyexc  endp

;****************************************
        include \c86\epilogue.h
        end
```